Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 1 (2012) 1527–1530

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Computational Science, ICCS 2010

Position paper on the design of a modular toolbox for the simulation of solid state lasers

J. Werner[a,b,*], M. Wohlmuth[a,b], C. Pflaum[a,b]

[a] *Universität Erlangen-Nürnberg*
Lehrstuhl für Systemsimulation (Informatik 10)
Cauerstraße 6, 91058 Erlangen, Germany
[b] *Erlangen Graduate School in Advanced Optical Technologies*

Abstract

In this work we describe the design of our simulation tool for solid state lasers. The software is based on modules to simplify several complex processes. Primarily, the adaptation of laser physics (e.g. thermal lensing, optical wave) with the appropriate simulation technique. Another aspect is to provide the possibility of exchanging the applied calculation methods or even to change the representation of the computational domain on demand.

Keywords: solid state laser, simulation, expression templates, unit tests, modular programming

1. Introduction

Designing a laser requires to take into account various competing physical effects like thermal lensing, focus shifts and gain dynamics. For complex systems, building a laser and measuring such an reference system is difficult. This approach would be costly and some effects, like thermal stress, are not measurable in acceptable quality. Often, it is not clear which physical property is most significant for a particular effect. Therefore a laser simulation tool has to be able to combine the prediction of physical influences with the capability to compare the simulation results with availeable experimental data.

2. Requirements

In order to provide a complete tool to simulate lasers, one has to take into account multiple needs.

Laser physics is a field of active research and produces new physical effects worth simulating. Thus one main task is to provide a possibility to implement additional computation techniques easily. Another one is to provide means to integrate these methods in a way they can reuse structures and algorithms already implemented. A further requirement is to be able to handle a wide range of materials, for example gain crystals, lens materials and cooling media. The

*Corresponding author
Email addresses: johannes.werner@informatik.uni-erlangen.de (J. Werner), matthias.wohlmuth@informatik.uni-erlangen.de (M. Wohlmuth), pflaum@informatik.uni-erlangen.de (C. Pflaum)

1877-0509
doi:10.1016/j.procs.2010.04.170

last aspect of the simulatory part is to provide the ability to test the correctness of the implemented techniques and models. To this end one needs to be able to feed the modules with actual measurements as well as generic data.

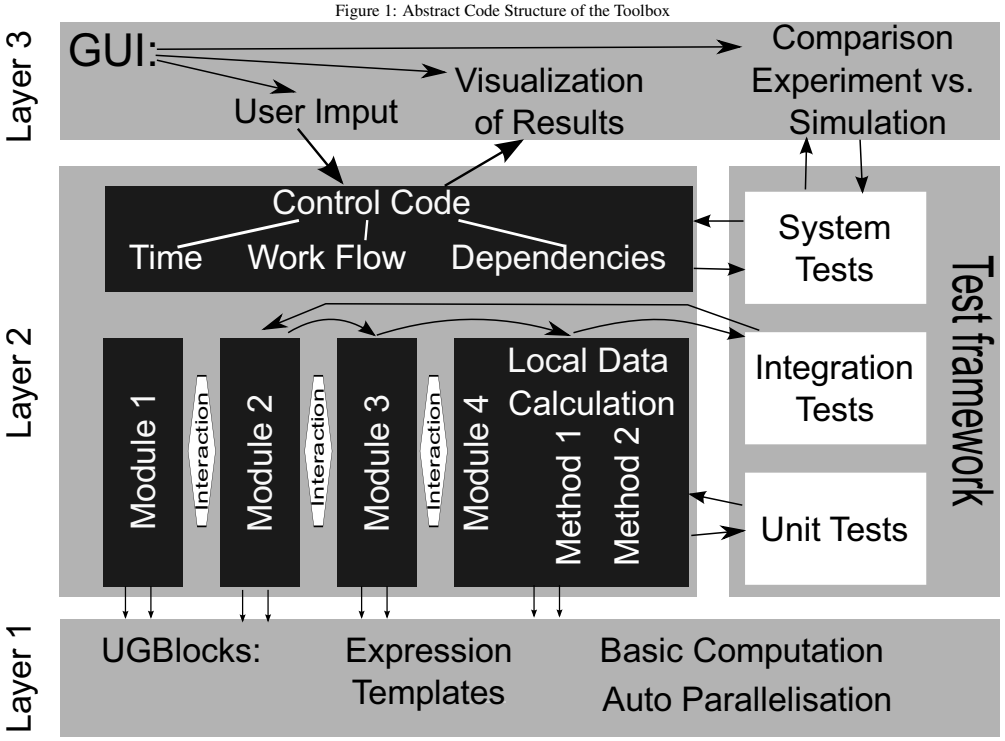
Additional there are some more general needs. The tool should be configurable and usable by a common laser developer. Also typical simulation setups should have decent runtime. Finally the user should be assisted in the choice of suitable simulation parameters, such as the number of necessary grid points or a suitable calculation technique.

3. Code Structure

In order to fulfill the above requirements, the code is divided into several parts which can be grouped in three layers.

The first layer builds the basis for computations. Its task is to handle all computational needs like data structures (e.g. grids) and basic numerical operations (e.g. stencils). By this, we remove the need to take care of implementory techniques within the implementation of the physical models. The implementation of the mathematical formulae is done by expression templates. This provides both, a way to handle multiple data sources and to feature a method of checking correctness of the implementation of the mathematical modelling at compile time. Further advantages of this implementation technique are its abilities to automatically parallelize the calculations or to transparently pass calculations to external libraries.

Figure 1: Abstract Code Structure of the Toolbox



On the second Layer the implementations of the physical simulation models and testing aspects happen, divided in three parts: The first part consists of a collection of modules that handle various simulations (such as pumping, heat distribution, deformation, laser modes, etc.). Those modules interact by providing calculation results, loading

simulation data or output results. Each module handles a specific task and may provide several ways of calculating the desired result. A second part provides a control code to manage the modules and to ensure proper interaction, as well as a low level user interface. Thus, these two parts provide the simulation core of the tool.

The third part of the software is a testing backend based on CppUnit (compare [1]) that supports automated tests on the basis of unit, integration and system testing (see Chapter 5 for further details).

Finally, the third layer is a graphical user interface to feed the simulation core with data and to visualize attained results.

4. Flexibility

Because of the separation of the simulation core from the implementation of the numeric part, modules to simulate laser effects can easily be added to the core. Also, within a module it is easy to change the used algorithm or data representation in order to adapt the code to specific simulation settings like resonator structures, crystal geometries or ray paths.

Furthermore, by providing the user with the ability to choose which modules are active, it is guarantied that the amount of unnecessary calculations is kept to a minimum.

5. Testing

Since the goal of any good simulation tool is to reflect the reality of physical effects in the calculated results, one needs to add a possibility to check every part of the code. The code has to be consistent with a reference model and also with real-life measured data. To achieve this, the testing environment is structured into several parts.

In order to check the code for correct adaptation of the mathematical model, we use two different approaches. First, we check by standard run time tests with generic data against known results. Second, we check the soundness of the implemented mathematical formula at compile time by using expression templates (see [2] and [3]).

In a second stage, the interaction between modules is checked. This includes testing their interfaces against the specification and running integration tests by loading prepared data into the modules and then running the calculation part of the collecting module. As with the mathematical checking, the result data can be compared to precalculated results.

As a last step, system tests are applied to simple configurations and compared to known results. The known results are received from other simulation software, direct calculations and measured data. In more complex settings, the results have to be compared with experimental data (see [4]).

6. Status

At the current stage various mathematical and simulation parts (on layer two) are implemented and working (see [5]). In the numerical handler remaining tasks involve speeding up utilizing auto parallelisation and providing thread safety on the interfaces to the driving codes in the core's modules.

The simulation modules are implemented in accordance with the current state of research and already used in laser simulations (see [6] and [4]).

In the testing environment the unit tests of serveral modules are already implemented and usable. For those modules integration testing is also available. It would be beneficial to automate the testing against measurements, which requires much user experience though. Therefore, the focus is on providing tools, like for example a visualization frontend, to assist manual checking.

7. Conclusion

We presented a comprehensive view on the design of our laser simulation toolbox. It accomplishes the primary goal of providing a development aid to the laser engineer. Furthermore, it guides the developers during the implementation of additional components and in ensuring the quality of the simulation results.

Acknowledgment

We gratefully acknowledge funding of Erlangen Graduate School in Advanced Optical Technologies (SAOT) by the German National Science Foundation (DFG) in the framework of the excellence initiative.

[1] P. Hamill, Unit Test Frameworks, O'Reilly Media, 2004.
[2] J. Härdtlein, C. Pflaum, A. Linke, C. H. Wolters, Advanced expression templates programming, Computing and Visualization in Science 12 (2009) 59–68. doi:10.1007/s00791-009-0128-2.
[3] C.Pflaum, Z. Rahimi, Automatic parallelization of staggered grid codes with expression templates, International Journal on Computational Science and Engineering 4 (4) (2009) 306–313. doi:10.1504/IJCSE.2009.029166.
[4] M. Wohlmuth, C. Pflaum, K. Altmann, M. Paster, C. Hahn, Dynamic multimode analysis of Q-switched solid state laser cavities, Optics Express 17 (20) (2009) 17303–17316.
[5] M. Wohlmuth, K. Altmann, C. Pflaum, Finite Element Simulation of Solid State Laser Resonators, in: Proceedings of SPIE, Vol. 7194, SPIE, SPIE, 2009, pp. 16–1 – 16–10.
[6] C. Pflaum, D. Seider, K. Altmann, Three-Dimensional Finite Element Computation of Laser Cavity Eigenmodes, Applied Optics 43 (9) (2004) 1892.